Modular Session Types for Distributed Object-Oriented Programming

Simon J. Gay
Nils Gesbert
University of Glasgow

António Ravara
New University of Lisbon
Formerly Technical University of Lisbon

Vasco T. Vasconcelos
Alexandre Z. Caldeira
University of Lisbon

POPL, 21st January 2010

Running Example: Mail Reader

```
enum ErrorStatus { OK, ERR }

interface MailReader {
  ErrorStatus login (String user, String pass);
  int getNumberOfMessages();
  ErrorStatus fetchAndDelete (int index);
  String getMessageContent();
  void logout();
}
```

Constraints

- Cannot fetch messages before (successful) login or after logout
- Cannot get the content of a message before it is fetched successfully
- *Should* not fetch a new message before the previous one is read or saved

Constraints

- Cannot fetch messages before (successful) login or after logout
- Cannot get the content of a message before it is fetched successfully
- *Should* not fetch a new message before the previous one is read or saved

Not any method can be safely called at any time; the interface (set of available methods) changes depending of what is done.
We propose to represent that fact at the type level.

Session Types for Objects

- Several methods available: external choice
  `{fetchAndDelete: S, logout: S'}`

  Object branches / Client selects by calling a method

Session Types for Objects

- Several methods available: external choice
  `{fetchAndDelete: S, logout: S'}`

  Object branches / Client selects by calling a method

- Dependency on a method result: internal choice
  `<OK: S, ERR: S'>`

  Object selects by returning a label / Client branches

Session Types for Objects

- Several methods available: external choice
  `{fetchAndDelete: S, logout: S'}`

  Object branches / Client selects by calling a method

- Dependency on a method result: internal choice
  `<OK: S, ERR: S'>`

  Object selects by returning a label / Client branches

Session `Init = {login: <OK: NoMsg, ERR: Init>}`

where `NoMsg = {fetchAndDelete: <OK: MsgRead, ERR: NoMsg>, getNumberOfMessages: NoMsg, logout: {} }`

and `MsgRead = {getMessageContent: NoMsg, getNumberOfMessages: MsgRead, logout: {getMessageContent: {}} }`

How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it

How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it
- Objects are linear but may be stored in fields of other objects

How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it
- Objects are linear but may be stored in fields of other objects
- External type of an object: session type, $C[S]$

How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it
- Objects are linear but may be stored in fields of other objects
- External type of an object: session type, $C[S]$
- Internal state of an object: type of its fields, $C[F]$

How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it
- Objects are linear but may be stored in fields of other objects
- External type of an object: session type, $C[S]$
- Internal state of an object: type of its fields, $C[F]$

Judgements:

- Expressions: $\Gamma \rhd e : T \lhd \Gamma'$
- For a method body: $\texttt{this} : C[F] \rhd e : T \lhd \texttt{this} : C[F']$
- Internal/External state compatibility: $F \vdash C : S$
  Coinductively checks method bodies in order

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`
  - Choice: $\oplus${`validate:` Σ, `cancel:` Σ'}
    Allows a selection by sending one of the labels

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`
  - Choice: $\oplus${`validate:` Σ, `cancel:` Σ'}
    Allows a selection by sending one of the labels
  - Branching: `&{ok:` Σ, `error:` Σ'`}`
    May receive any of the labels

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`
  - Choice: $\oplus$`{validate:` Σ`, cancel:` Σ'`}`
    Allows a selection by sending one of the labels
  - Branching: `&{ok:` Σ`, error:` Σ'`}`
    May receive any of the labels

Channels can be treated like objects

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`
  - Choice: $\oplus$`{validate: `Σ`, cancel: `Σ'`}`
    Allows a selection by sending one of the labels
  - Branching: `&{ok: `Σ`, error: `Σ'`}`
    May receive any of the labels

Channels can be treated like objects

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$
$$[\![\&\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{receive}_E : \langle l : [\![\Sigma_l]\!]\rangle_{l\in E}\}$$

Session Types for Channels

- Honda et al., 1993-present
- Originally meant for typing communication channels in the π-calculus: describes a protocol
  - Sequence, send, receive: `!String.?Bool...`
  - Choice: $\oplus$`{validate: `Σ`, cancel: `Σ'`}`
    Allows a selection by sending one of the labels
  - Branching: `&{ok: `Σ`, error: `Σ'`}`
    May receive any of the labels

Channels can be treated like objects

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$
$$[\![\&\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{receive}_E : \langle l : [\![\Sigma_l]\!]\rangle_{l\in E}\}$$
$$[\![\oplus\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{send}_l : [\![\Sigma_l]\!]\}_{l\in E}$$

Example: simplified POP3 Protocol

Type POP3 = !String.!String.&{OK: Trans, ERR: POP3}

where Trans =

```
  ⊕{
    STAT: ?int.Trans,
    DELE: !int.Trans,
    QUIT: End,
    RETR: !int.&{OK: ?String.Trans, ERR: Trans}
  }
```

A POP3 Client Class (implements MailReader)

```
class POP3Client implements MailReader {
  Chan[POP3] c; Null n, m;                                  // fields
  ErrorStatus login (String user, String pass) {
    c.send (user); c.send (pass);
    switch (c.receive()) {
      case OK: c.send (STAT); n = c.receive(); return OK;
      case ERR: return ERR;
  }}
  ErrorStatus fetchAndDelete (int index) {
    c.send (RETR); c.send (index);
    switch (c.receive()) {
      case OK: m = c.receive(); c.send (DELE); return OK;
      case ERR: return ERR;
  }}
  int getNumberOfMessages() {return n;}
  String getMessageContent() {return m;}
  void logout() {c.send (QUIT);}
}
```

Relation between internal and external states

$$\texttt{\{Chan[POP3] c; Null n; Null m\}} \vdash \texttt{POP3Client : Init}$$

Relation between internal and external states

```
{Chan[POP3] c; Null n; Null m} ⊢  POP3Client : Init

 <OK: {Chan[Trans] c; int n; Null m},
  ERR: {Chan[POP3] c; Null n; Null m}>
⊢
  POP3Client : <OK: NoMsg, ERR: Init>
```

Relation between internal and external states

```
{Chan[POP3] c; Null n; Null m} ⊢  POP3Client : Init
```

```
 <OK: {Chan[Trans] c; int n; Null m},
  ERR: {Chan[POP3] c; Null n; Null m}>
⊢
  POP3Client : <OK: NoMsg, ERR: Init>
```

```
 {Chan[Trans] c; int n; Null m} ⊢  POP3Client :  NoMsg
```

etc.

Subtyping

- An object with more methods can be safely used in place of an object with less methods

  e.g. add fetch(); add getNumberOfMessages() at the end

- An object with less internal choice (more deterministic) can be safely used in place of an object with more internal choice

Implementation: Bica

A type-checker has been implemented on top of Java, using Polyglot: Bica

Implements several extensions wrt the formal system:

- Shared objects along with linear ones
- While loops
- Inheritance

Standard Java semantics. Uses Java annotations for session types.

http://gloss.di.fc.ul.pt/bica/

Conclusion

Results:

- Type safety:
  - Conformance: sequence of method calls on an object always follows its declared session
  - Communication safety in a distributed setting
- Typechecking algorithm and prototype implementation

Conclusion

Results:

- Type safety:
  - Conformance: sequence of method calls on an object always follows its declared session
  - Communication safety in a distributed setting
- Typechecking algorithm and prototype implementation

Related to (mainly):

- Session types for OO languages: Dezani-Ciancaglini et al.
- Typestates for objects: Bierhoff and Aldrich

Conclusion

Results:

- Type safety:
  - Conformance: sequence of method calls on an object always follows its declared session
  - Communication safety in a distributed setting
- Typechecking algorithm and prototype implementation

Related to (mainly):

- Session types for OO languages: Dezani-Ciancaglini et al.
- Typestates for objects: Bierhoff and Aldrich

Missing: Aliasing control

- Can we use existing work?
- How does it relate with subtyping?

Conclusion

Results:

- Type safety:
  - Conformance: sequence of method calls on an object always follows its declared session
  - Communication safety in a distributed setting
- Typechecking algorithm and prototype implementation

Related to (mainly):

- Session types for OO languages: Dezani-Ciancaglini et al.
- Typestates for objects: Bierhoff and Aldrich

Missing: Aliasing control

- Can we use existing work?
- How does it relate with subtyping?

Full version with proofs: technical report available at http://gloss.di.fc.ul.pt/bica/